Geometric Displacement on Plane and Sphere

Elodie Fourquet* William Cowan† Stephen Mann‡

David R. Cheriton School of Computer Science
University of Waterloo

ABSTRACT

This paper describes a new algorithm for geometric displacement mapping. Its key idea is that all occluded solutions for an eye ray lie in two-dimensional manifolds perpendicular to the underlying surface to which the height map is applied. The manifold depends only on the eye position and surface geometry, and not on the height field. A simple stepping algorithm, moving along the surface within a manifold renders a curve of pixels to the view plane, which reduces height map rendering to a set of one-dimensional computations that can be done in parallel. The curves on the view plane for two specific underlying manifolds, a plane and a sphere, are straight lines. In this paper we focus on the specific geometry of simple underlying surfaces for which the geometry is more intuitive and the sampling of the rendered image direct.

Index Terms: I.3.3 [Computer Graphics]: Picture/Image Generation—Viewing Algorithms

1 INTRODUCTION

Texture mapping is an extremely successful modelling and rendering technique. We attribute this success to five reasons.

1. Texture mapping is inherently two-dimensional. Mapping a two dimensional texture to a two-dimensional manifold is easy for modellers to comprehend and for programmers and hardware designers to implement.

2. Texture mapping separates potentially complex rendering of fine detail into two parts. The first part is creating the texture. Computationally expensive texture creation can be done off-line, which is essential for photographing or scanning real world content and for drawn or painted content. The second part is rendering the texture, which must be done on-line.

3. Textures give an appearance of three-dimensional detail, while requiring geometric calculations that are simple. The underlying surface, to which the texture is mapped, is geometrically simple compared to the geometry the texture approximates.

4. Texture rendering algorithms evaluate each pixel independently of the others, and are easy to parallelize.

5. And most important, texture mapping gives visually convincing results in many applications.

Advantage 1 is particularly important because it makes rendering textures independent of the content of the texture. These algorithms are, in fact, rare examples of graphics algorithms that are truly real-time, because they depend only on the number of texture pixels on the view screen and on the number of texels in the texture, both of which are bounded above.

*e-mail: efourque@cgl.uwaterloo.ca
†e-mail:wmcowan@cgl.uwaterloo.ca
‡e-mail:smann@uwaterloo.ca

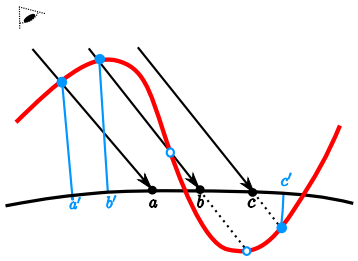


Figure 1: Vertical section of a height map (red curve) applied to an underlying surface (black curve). Three eye rays (arrows) are shown. Two of them intersect the surface at a and c, and the height map at a' and c'. The third eye ray intersects the surface at b, but intersects the height map three times, b' being the desired intersection.

The success of texture mapping quickly led to modifications and extensions having similar advantages. Blinn introduced bump mapping [2], and Max fully exploited it to give an appearance of three dimensionality in an animated scene by locating highlights correctly as lighting and viewpoint change, while retaining all the advantages of texture mapping [10]. Displacement mapping [3], also known as height mapping, improves bump mapping to include self-occlusion within the texture, and correct silhouettes at the boundaries of the texture. In doing so, however, displacement mapping compromises some of the advantages of texture mapping.

This paper revisits the geometry of displacement mapping for two simple underlying surfaces, mapping normal and height values at each point and with respect to the eye and view plane location. The objectives of reanalyzing the geometry is to find a mapping algorithm for displacement that preserves as many of the identified advantages of texture mapping as possible. The basic geometry of displacement mapping is simple: the texture defines a surface colour, a surface normal and a height over a finite domain. The texture is applied to a finite region of an underlying surface, a section of which is shown in Figure 1. We calculate the intersection of eye rays with the underlying surface, (a and c), from which we derive the intersection of the eye ray with the height field (a' and c'), which is usually offset from the intersection of the eye ray with the surface. The colour and texture normal at the intersection, projected to the underlying surface along the surface normal, are used for calculating the colour on the particular eye ray.

The principle is simple to understand, but intersections with the displacement map are not easy to find in practice, especially when an eye ray intersects an height field more than once, as does the middle ray in Figure 1. The ray intersects the underlying surface at b but intersects the height map three times. The closest non-occluded intersection, b', is the desired solution. These cases are particularly important because they are the source of self-occlusion, an important visual cue to the geometry of the surface.

Of course, finding the correct solution can be achieved by constructing the height-mapped surface as a three-dimensional object,

and applying three dimensional ray-surface intersection, choosing the intersection closest to the eye. As discussed in Section 2, many current algorithms rely on this approach, taking advantage of the GPU to render at interactive rates. But, being inherently three dimensional, these GPU methods lose the advantage 1 of texture mapping, and sometimes advantage 3, while retaining advantage 4. Other algorithms that guarantee the correct solution are incremental, retaining advantages 1 and 3 but losing advantage 4. Our goal is to study the geometry of displacement mapping for two particular surfaces, presented in Section 3, to understand better the opportunities and pitfalls in a novel geometrical approach to the height map intersection problem.

Our observation is that the displacement map intersection problem can be solved as an ordered one dimensional problem on curves passing through a particular point. Coherent stepping and sampling provide the non-occluded intersection solution without making the problem three dimensional, while remaining solvable in parallel along different curves. To exploit this observation the following three elements are needed:

- a stepping algorithm along the height map that finds the non-occluded intersection,
- a mapping between samples on the view plane and their corresponding points on the underlying surface, and
- a method to create a sampling pattern on curves in the view plane that has uniform planar density.

Our examination of geometry leads to a natural algorithm, which is the base case for all algorithms that step along the height map. This algorithm is described in Section 4 with diagrams. An implementation with the mathematical details for sampling and for mappings between the view plane and both underlying surfaces, the plane and the sphere, are described in Section 5. Result images are presented and performance discussed in Section 6.

2 Rendering Algorithms for Displacement Maps

Displacement mapping was first suggested over two decades ago [3] [12], but suffered from two drawbacks. Authoring content for displacement mapping, by algorithm or artist, was costly. Mapping algorithms ran slowly, and without good content there was little incentive to improve them. The advent of commercial 3D scanners around 2000 solved the first problem. About the same time GPUs began to surpass CPUs in terms of computing throughput.

Suddenly, a surfeit of real world height maps were available for mapping, and GPUs with sufficient power to do many rendering operations became available. Soon, GPU displacement mapping algorithms appeared [6]. GPU capabilities have increased rapidly in the last few years; at present they are able to ray-cast fairly complex scenes in real-time. As a result ray-casting is the preferred method for rendering displacement maps [13]. Ray casting on GPUs is usually done by stepping along the eye ray, and exhibits the usual problems of stepping algorithms [14]. Currently the state of the art in GPU ray casting for displacement mapping uses sphere tracing [4], an algorithm created for speeding up CPU intersection tests with implicit surfaces [5]. This interaction between CPU and GPU algorithms reinforces our belief that taking a closer look at displacement map geometry will yield benefits.

To get better performance than simple ray-casting, some researchers are currently looking at multi-scale models of surfaces in which ray-casting is performed hierarchically at progressively finer levels of detail [11] [15]. This approach, in which displacement maps are applied recursively to larger scale displacement maps emphasizes the importance of displacement maps for modelling as a generalized form of surface pasting [8].

Recent GPU algorithms, which are restricted in their manipulation of geometry, have eclipsed earlier displacement mapping research, which focussed on a better understanding of geometry as a way of making more efficient displacement mapping algorithms. The most geometry oriented of that earlier research was in terrain mapping [7], which scanned the terrain along eye rays projected onto a plane, a special case of the visibility curves we develop in this paper. Their stepping algorithm is different from ours, moving along the underlying plane, and not along the displacement map. This choice gives them less flexibility in how to handle complex silhouettes. In effect, an eye ray that does not intersect the underlying surface may well intersect the displacement map especially close to the boundaries of the map, or when tall features are present on the map and the scene is viewed from a low angle. By stepping along the underlying surface, those details at the boundary edges are missed and most surface stepping algorithms fill those pixels with pseudo-data.

Thus, in terms of the current displacement mapping research the work reported in this paper is a return to the past. However, we regard that as a virtue, because it reexamines the geometric foundation on which current algorithms are based, the best opportunity for discovering a different style of algorithm.

3 Geometry

This paper presents a new approach to the geometry of displacement maps on planes and spheres. Those surfaces are simple, but demonstrate fully how to take advantage of spatial coherence in the height field. This geometry is exploited in the stepping algorithm described in Section 4. Our algorithm steps along visibility curves defined by the eye position and the underlying surface as a way of dealing effectively with multiple solutions. This section describes the geometric basis of the visibility curves, as illustrated in Figures 2 and 3.

3.1 Terminology

We use the following terminology in the description of the geometry of displacement map and our stepping algorithm. A *height field* is a single-valued function $h(s,t)$ that for any point (s,t) in its domain gives the height above that point. The height map is applied to a finite region of the *underlying surface*, f, which may itself be infinite. The underlying surface has at each point a normal vector, $\hat{n}_s(s,t)$, called the *surface normal*, which is determined by the geometry of the underlying surface. To apply the height field to the underlying surface at (s,t), we *displace* from the underlying surface, along the surface normal, by an amount equal to the height field.

In world coordinates, the point (s,t) on the surface is $O+f(s,t)$, where O is the origin of world coordinates. The *eye ray* is $O+\alpha\hat{v}$, where $\alpha \geq 0$ and the eye is placed, without loss of generality, at the origin of world coordinates. When we say that the eye ray intersects the height field at (s',t') we mean that the intersection occurs at $O + f(s',t') + h(s',t')\,\hat{n}(s',t')$. It is important to note that an eye ray intersecting the surface at (s,t) usually intersects the height map at a different point (s',t'), for which we must solve. The generalized *texture map* defines at each point on the surface a colour, $C(s,t)$, a normal vector, $\hat{n}_t(s,t)$, called the *texture normal*, and the height, $h(s,t)$.

The *centers of projection*, CoP are important points linking the eye position to the underlying surface. They are points on the underlying surface where the normal, $\hat{n}_s(s,t)$, is parallel to their eye ray. For the plane there is a single CoP which is the point on the surface closest to the eye. For the sphere there are two, the closest and the farthest points from the eye, of which one the nearest, is non-occluded. Thus, CoP_s defines the closest point on the underlying surface to the eye, and CoP_v defines that point projected to the view plane.

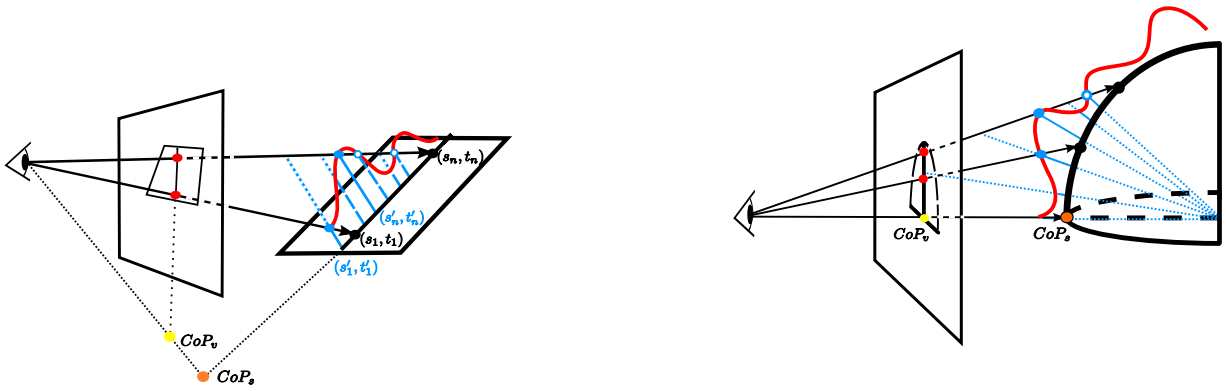


Figure 2: Geometry for a particular cross section of the height map on a plane and a sphere. The CoP_s is the point on the underlying surface closest to the eye. A line passing through that point is the intersection of a cross section plane including the eye, with the underlying surface. Eye rays on this cross section plane intersect the height field in a sequential manner, from closest to the CoP_s to away from it.

3.2 Problem Statement and Approach

Given an underlying surface of a plane or a sphere onto which a height field is applied, and given an eye position and a view plane, our goal is to solve the eye ray and height field intersection problem in one dimension. Our approach is to search along lines in the view plane. For an arbitrary line in the view plane this requires a two dimensional search, but in this section we show that there are special lines in the view plane for which search is one dimensional. We call these lines visibility lines in the view plane.

We must determine where each eye ray intersects the height field, specifically the point on the underlying surface from which the intersection is displaced. The two-dimensional mapping between a point on the view plane and the corresponding point on the underlying surface, from the red points to the black ones in Figure 2, is straightforward. However, the view plane point should have the colour, and normal of the non-occluded intersection of the eye ray with the height map, the solid blue points in Figure 2. Finding this intersection is challenging. In effect, depending on the angle of the eye ray and the geometry of the height field, multiple intersections are possible, as shown in Figure 2, in which case we must find the non-occluded solution.

The geometry of the eye ray, underlying surface and height map has some hitherto unnoticed regularities that make finding the intersection easier. For underlying surfaces that are planes or spheres all intersections of an eye ray with any height map, including all multiple intersections, lie on a plane, P, spanned by the eye ray and the normal at the surface intersection, as shown in Figure 2. Further, we now have a one dimensional slice of our height map: the intersection of P with the underlying surface, S, gives a curve C, and the height field of S over C lies in the plane P, as illustrated in Figure 2. We can then group eye rays into ordered sets such that the non-occluded intersection on any ray is the solution closest to the solution of the previous eye ray in the set, finding solutions sequentially, and using spatial coherence in the height field to find each next solution easily.

Each set of rays intersects the underlying surface along a *visibility curve*. The visibility curves cover the surface and share a common point, the CoP_s, the point on the surface closest to the eye. While eye rays must be processed in order along each curve, the curves can be processed in any order.

Eye rays are processed sequentially along each curve. But which direction guarantees finding the non-occluded intersection when multiple intersections exist? For any eye ray the non-occluded intersection is the one closest to the eye. The eye ray to the closest point on the surface, CoP_s, intersects the height field at a single point when the CoP_s is within the mapped region of the defined underlying surface. Since all visibility curves share the CoP_s we can start the algorithm there for every curve. Indeed, by starting to solve near the closest CoP_s and stepping away from it along each visibility curve, then the solution closest to the previous solution is always the non-occluded one. Thus, a stepping algorithm that finds the first solution correctly can return the closest non-occluded intersection along the curve for the remainder eye rays in the set. Then, to complete the geometry we need to find the form of the visibility curves for the underlying surface.

3.3 Visibility Curves for Planes and Spheres

The visibility curves for an underlying surface are determined by the surface normals and the eye position. Figure 3 show the visibility curves for a plane and a sphere.

For planes, there is a single CoP_s which is the point where the eye ray is perpendicular to the plane. CoP_v is the projection of this point to the view plane. (The geometry is not well-defined in a single negligible case, when the eye point lies on the underlying plane.) Because all points on a plane have the same surface normal, points that displace to the same eye ray lie on a straight line, and the set of all visibility curves is a pencil of lines, centred on the CoP_s. This pencil maps to the view plane as a pencil of lines centred on the CoP_v. Thus, when the underlying surface is a plane, the visibility curves have their simplest form, lines on the plane radiating from the CoP_s, which project to lines on the view plane radiating from the CoP_v. If the CoP_s lies within the height map then the eye rays are processed outward from the CoP_s. If it does not then they are processed from the eye ray closest to the CoP_s, proceeding away from the CoP_s.

On a sphere the geometry is more complex, but similar. There are two centers of projection, a minimum closest to the eye, the north pole, and a maximum farthest from it, the south pole. (The terms ‘north’ and ‘south’ have, of course, no significance with respect to world coordinates. They merely provide a convenient way of describing the curves. As we orient the ‘north’ of the sphere to face the eye, only the northern hemisphere is visible, and not quite

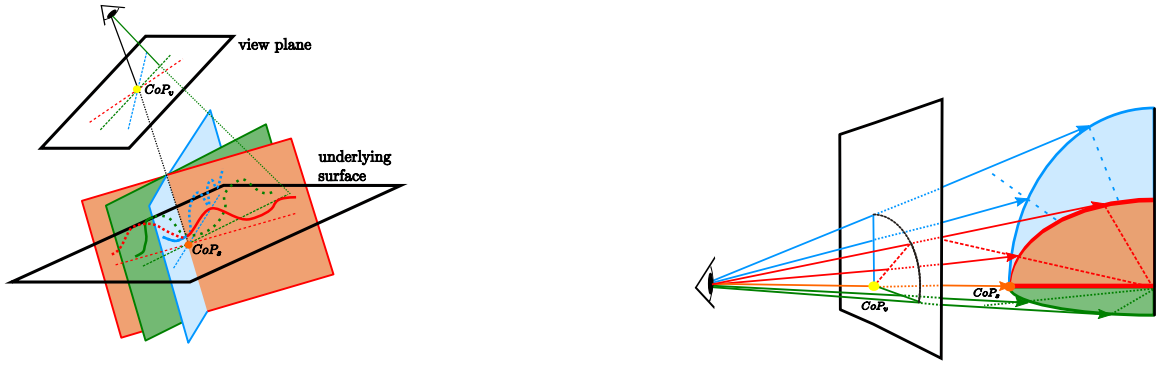


Figure 3: The left diagram shows that the coherent visible curves for an underlying plane are a pencil of lines on both the surface and the view plane, intersecting respectively at CoP_s and CoP_v. The right diagram shows that the coherent visible curves for an underlying sphere are the set of great circles intersecting at the CoP_s for the surface, which maps to a pencil of lines on the view plane meeting at the CoP_v.

all of it because the distance to the eye is finite. The whole southern hemisphere is back-facing.)

The visibility curves are great circles passing through north and south poles: that is, lines of constant longitude. To see why, notice that all normal vectors on a sphere point radially away from the sphere centre. A great circle is the intersection of the sphere with a plane passing through the centre of a sphere, the equator being an example. Thus, the normal vectors on a great circle lie in a plane. Furthermore, because the great circles pass through the north pole, the minimum CoP_s lies on each of these planes. For each eye ray to the sphere, the set of possible intersections with the height field, lies on the curve defined by the points whose surface normal, $\hat{n}_s$, displaces to the ray. Thus, the set of points on the same visibility curve on the sphere are on a same plane that contain the sphere center. Therefore, every eye ray is part of a cutting plane that intersects the sphere on a great circle, on which lies the set of points that displace to possible intersection solutions with the height map.

Because this set of eye rays lies in a plane, they intersect the view plane on a line passing through the projection of the north pole, CoP_v. On the view plane they are a pencil of straight lines centred on the CoP_v, the pattern we found for the plane, though originating in a different geometry.

The pattern of visibility curves on the view plane, a pencil of lines centred on the CoP_v, also occurs in multicamera views of a scene, because the geometry is similar. It has been used for calculating occlusion [9] because it shares the ordering properties exploited below.

4 The Stepping Algorithm

The geometry discussed above suggests a natural algorithm. An eye ray terminating on a plane or a sphere defines a cutting plane, which contains a visibility curve, all surface normals on the curve, all eye rays that terminate on the curve, and all solutions where an eye ray intersects a height field. Figure 1 shows a cross-section in such a cutting plane.

The computation on each cutting plane is independent of the computations on other planes. Therefore, we have reduced a 2D computation on a surface to a collection of independent 1D computations, one for each visibility curve. In each 1D computation, the solution for one eye ray depends on the solutions of previous eye rays, which makes the computation less distributable than bump or texture mapping, but which takes advantage of spatial coherence in the height field.

A natural algorithm results from two observations. First, when a solution is known, spatial coherence makes it easy to find the next solution by stepping along the height field. And, second, when an eye ray intersects the height field in several places, the non-occluded solution is the first found. Thus, the natural algorithm steps along the height field, finding the next solution reliably provided only that we start stepping at a good enough solution.

In this section, the underlying surface in the figures is a plane because the diagrams are easier to understand. Equivalent figures for a sphere use horizontal steps that follow the curvature of the surface, with normals that are perpendicular to the tangent at the surface point. Section 5 gives the mathematics for each surface in detail. In addition, the eye point is always further from the surface than the one drawn. Finally, the distance between successive eye rays, Δs, is exaggerated to show the details of stepping more clearly. Spatial coherence is greater in practice, because the eye rays are closer together compared to the surface detail.

4.1 Step

The algorithm steps along each visibility curve from one eye ray to the next, following the height map. Eye ray positions are determined by sample points on the view plane, placed so that each pixel gets about the same number of samples as any other. The details of the sampling method are described in Section 5.3. Two consecutive eye rays are separated on the underlying surface by a distance Δs. The heart of the algorithm is stepping along the height field so that, given the solution for one eye ray, we find the solution for the next one. A horizontal step, to intersect the eye ray, can be in either direction: forwards (advance step) or backwards (reverse step) along the visibility curve.

4.1.1 Advance Step

In Figure 4, the left diagram shows the stepping algorithm. The solution a' for the left eye ray, which intersects the underlying surface at a, is known. The solution b' for the next eye ray, which intersects the underlying surface at b, is to be found. From the known solution a horizontal step to the next eye ray, Δd_1, is taken, followed by a vertical step to the height field. Horizontal and vertical steps alternate until we are 'close enough' to the solution. The solution illustrated converges at b' after three iterations. The algorithm's convergence condition is that the length of a horizontal step Δd_n falls below a pre-determined threshold ϵ, which is a fraction of the sampling distance Δs. That is, the threshold is $\epsilon \Delta_s$.

4.1.2 Reverse Step

To find the next solution, horizontal steps are taken parallel to the surface. It is possible to overstep, passing the solution. Then the

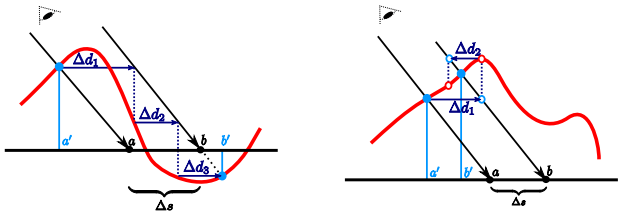


Figure 4: On the left typical iteration of advance steps. On the right, intersections with the height map and the eye ray that occur for a reverse step.

next step to the eye ray reverses direction as shown in the right diagram of Figure 4. Continued stepping may circle the solution and can even diverge. When a reverse step occurs the stepping algorithm uses binary search to interpolate between the ends of the reverse step until the interval containing the solution is smaller than the threshold.

4.1.3 Remarks

The stepping algorithm has several possible pathologies. The most common occurs when the height field is almost parallel to the eye ray. Then the number of steps needed to traverse the narrow space between them can increase without limit. This is a frequently-observed property of algorithms that find their solutions by stepping, such as hill-climbing algorithms. The pathology is, however, unimportant for displacement mapping, where there are several eye rays per pixel. Every intermediate solution lies within the pixel and it does not matter which one is obtained. Thus, a generous threshold stops stepping after a small number of steps without making the resulting image any less accurate. Most other pathologies are unimportant for similar reasons.

The stepping algorithm is superficially similar to earlier work [7], but there is an important difference. We step along the height field using the eye rays to size the steps, while Lee et al. step along the surface, using texel density to control the step size. The difference is most significant when there are many texels per eye ray, as occurs when the surface is distant or when the angle between the eye rays and the surface is small. For distant surfaces stepping along the height field has a big advantage; for low eye ray angles, however, stepping along the height field can suffer from over-stepping pathologies, which can leave holes in occluding surfaces. Such pathologies need to be remedied, usually by adjusting the length scale of the height field.

4.2 First Solution

Finding the next solution given a previous one is not enough: a first solution is needed. The general principle is that stepping moves from steeper eye rays to less steep ones, the direction of which ensures that occluding solutions precede occluded ones. If the CoP_s lies within the mapped region, stepping starts at the CoP_s and moves away from it along the visibility curve. At the CoP_s the eye ray is perpendicular to the underlying surface, and the solution occurs where the eye ray intersects the underlying surface, as shown in Figure 5. Stepping moves along the visibility curve away from the CoP_s using the algorithm described above.

More often, however, the CoP_s lies outside the mapped region as on the left in Figure 2. Then there is a point where the visibility curve enters the region, the *leading edge*, which is the point on the visibility curve closest to the CoP_s, such as (s_1, t_1) in the figure and a point where it leaves the region, the *trailing edge*. (s_n, t_n) in the figure is a point close to the trailing edge. At the leading edge the rays point into the region from outside it, as shown in the figure. In such cases we find the intersection of the edge of the

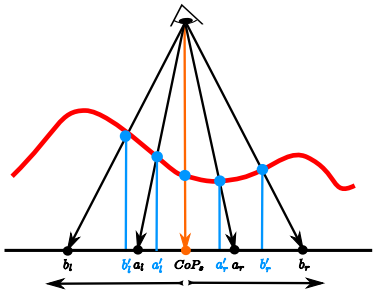


Figure 5: Eye rays near a CoP_s inside the underlying surface. The arrows pointing in opposite directions show the opposite directions taken along the visibility curve for the starting step on either side of the CoP_s.

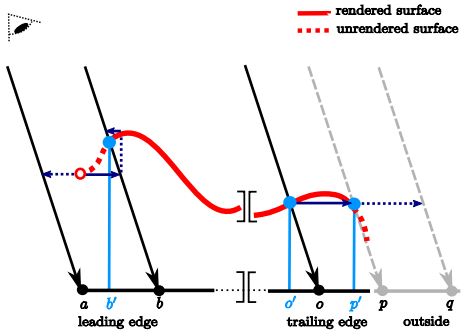


Figure 6: Boundary conditions of the algorithm for a CoP outside the underlying surface. The proper silhouette is rendered by not rendering some starting rays that intersect the underlying surface, whereas rendering some final ones that do not.

region that is closest to the CoP_s. We then choose the intersection of the visibility curve with the boundary of the region that is closest to the CoP_s as the starting point, as illustrated in Figure 6. Thus, the eye rays are guaranteed to be directed into the mapped region, as shown.

The leftmost eye ray intersects the underlying surface at a in Figure 6. Because there is no previous solution, the starting point is the intersection of the ray with the underlying surface, from which a vertical step is taken to the height field. The following horizontal step, a reverse, goes outside the mapped region. The height field is undefined and the ray is not rendered. The next eye ray, intersecting the surface at b, uses the height at a as the previous solution. In the figure, the second ray converges quickly to the correct solution, but if, like the first ray, it has no solution, the last of its height field intersections is taken as the starting point for the next ray. This process is essential for getting the leading edge silhouette correct.

4.3 Last Solution

To complete the stepping algorithm a stopping criterion is needed. A possible criterion might stop at the last eye ray intersecting the mapped region, but this criterion does not calculate silhouettes correctly, as shown in Figure 6. Eye ray o is the last to intersect the underlying surface. From its solution the stepping algorithm easily finds the solution for eye ray p, even though this eye ray does not intersect the inside of the underlying surface. Rendering this solution is essential for the correct silhouette, so it is unacceptable to stop when eye rays no longer intersect the surface. Notice, however, processing the next ray q steps outside the region where the height

field is undefined. This is the termination condition: stepping stops when a horizontal step goes outside the mapped region.

To consider all the cases, negative height field, i.e., under the underlying plane, or eye point below the horizon line, a bounding box may be useful to get the correct silhouette at the leading and trailing edges. For the sphere, a larger sphere, enclosing the entire height map, is used to get the silhouette at the trailing edge. In effect, eye rays are generated to include the maximum height value on the sphere profile. Also the equivalent of horizontal stepping on the sphere reaches outside where the step ends on southern hemisphere.

5 Implementation

Implementing the stepping algorithm requires computation of the visibility curves, cutting planes, on the underlying surface and on the view plane. Thus, we need easy to compute invertible mappings between sample points on the view plane their images on the underlying surface. Having restricted ourselves to simple underlying surfaces, it is possible to derive two dimensional mappings between the view plane and the plane or the sphere. The following two sub-sections give 3×3 matrices for the mapping for planes and non-linear equations for spheres.

Using the mappings between view plane and underlying surfaces we create an algorithm for uniform sampling along visibility lines on the view plane. Finally, a summary of our algorithm in the form of a pseudo-code completes this section.

For both the plane and the sphere we use a right-handed world coordinates, $(\hat{\imath}, \hat{\jmath}, \hat{k}, O)$ with both view plane and model down the negative z-axis and the eye placed, without loss of generality, at the origin. The view plane and underlying surfaces have internal coordinate frames defined in terms of the world coordinate frame. The view plane sphere transformations are done in more detail because spherical coordinates are less familiar than rectangular ones.

5.1 Plane

The view plane and the underlying surface are both planes, with respective coordinate frames, $(\hat{\imath}_v, \hat{\jmath}_v, O_v)$ and $(\hat{\imath}_s, \hat{\jmath}_s, O_s)$. On each plane, a point, P_v or P_s, is locally defined as a coordinate pair, (u, v) for the view plane and (s, t) for the underlying plane surface of the displacement map, using the following matrix notation.

$$P_v = u\hat{\imath}_v + v\hat{\jmath}_v + O_v = \begin{bmatrix} \hat{\imath}_v & \hat{\jmath}_v & O_v \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$\text{and } P_s = s\hat{\imath}_s + t\hat{\jmath}_s + O_s = \begin{bmatrix} \hat{\imath}_s & \hat{\jmath}_s & O_s \end{bmatrix} \begin{bmatrix} s \\ t \\ 1 \end{bmatrix}.$$

Using projective geometry we can map from local coordinates of one plane to another using an invertible 3×3 matrix. (The matrix is singular only in the uninteresting case where one plane maps to a point or line in the other plane.) The exact form of the matrix follows immediately from expressing the points in world coordinates. For the underlying plane, define S by

$$[\hat{\imath}_s \;\; \hat{\jmath}_s \;\; O_s - O] = \begin{bmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \end{bmatrix} \begin{bmatrix} \hat{\imath}_s \cdot \hat{\imath} & \hat{\jmath}_s \cdot \hat{\imath} & (O - O_s) \cdot \hat{\imath} \\ \hat{\imath}_s \cdot \hat{\jmath} & \hat{\jmath}_s \cdot \hat{\jmath} & (O - O_s) \cdot \hat{\jmath} \\ \hat{\imath}_s \cdot \hat{k} & \hat{\jmath}_s \cdot \hat{k} & (O - O_s) \cdot \hat{k} \end{bmatrix}$$
$$= \begin{bmatrix} \hat{\imath} & \hat{\imath} & \hat{k} \end{bmatrix} S,$$

with an equivalent definition for the view plane, V.

Because S and V are invertible, the mappings, $(u, v) \rightarrow (s, t)$ and $(s, t) \rightarrow (u, v)$ are well-defined.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \alpha V^{-1} S \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} \text{ and } \begin{bmatrix} s \\ t \\ 1 \end{bmatrix} = \alpha' S^{-1} V \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

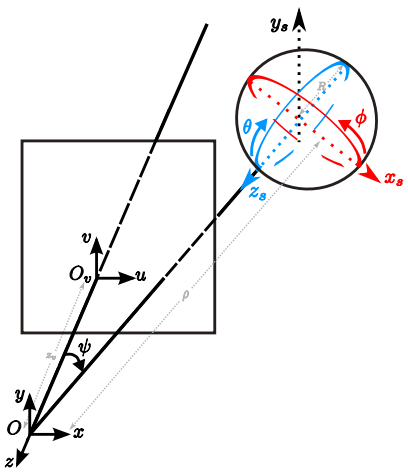


Figure 7: Geometry configuration used in the mathematics of the mapping between the visibility curves of the sphere and the lines on the view plane.

where the factors α and α' account for the normalizing division that occurs in projective transformations. $Q = V^{-1}S$, with $\alpha = 1/(q_{20}s + q_{21}t + q_{22})$ is then the matrix representing the projective transformation that maps from the surface to the view plane, and its inverse is $Q^{-1} = Q' = S^{-1}V$, with $\alpha' = 1/(q'_{20}u + q'_{21}v + q'_{22})$.

5.2 Sphere

For the case of the sphere, we derive the relation between a point (u, v) on the view plane and a point on the sphere defined in polar coordinates as (θ, ϕ). We consider a sphere of radius R with its center on the xz-plane. Its centre lies at a distance ρ from the eye, and at an angle ψ from the negative z-axis, as shown in Figure 7. Thus, the sphere center is at $(x_C,\ y_C,\ z_C) = (\rho \sin\psi, 0, -\rho\cos\psi)$.

A point on the sphere is labelled by coordinates (θ, ϕ), indicating its latitude (θ) and longitude (ϕ). A general point on the underlying surface is given by $\mathbf{f}(\theta, \phi) = (R\sin\theta\cos\phi, R\sin\theta\sin\phi, R\cos\theta)$. As spherical coordinates must be oriented, we select the most convenient orientation: the z-axis of the sphere ($\theta = 0$) pointing toward the eye, that is, toward the origin of world coordinates, and the zero of longitude ($\phi = 0$) lying in the xz-plane. The resulting orientation as defined is shown in Figure 7. Thus great circles of constant longitude, $\phi = constant$, map to straight lines in the view plane, as described in Section 3 and illustrated on the right diagram of Figure 3).

Lines of constant latitude, $\theta = constant$, are circles centred on and perpendicular to the line from the eye to the sphere centre. They have radius $R\sin\theta$. Any point on a circle is determined by its longitude ϕ. Thus, in world coordinates a point on the sphere is displaced from the sphere centre by $(x_p, y_p, z_p) = (R\sin\theta\cos\phi\cos\psi, R\sin\theta\sin\phi, R\sin\theta\cos\phi\sin\psi)$.

Therefore, in the world frame, an arbitrary point on the sphere, (θ, ϕ), has coordinates

$$\begin{aligned} (x_P &= \rho\sin\psi - R\cos\theta\sin\psi + R\sin\theta\cos\phi\cos\psi, \\ y_P &= R\sin\theta\sin\phi, \\ z_P &= -\rho\cos\psi + R\cos\theta\cos\psi + R\sin\theta\cos\phi\sin\psi). \end{aligned}$$

It projects onto the view plane, which is at $z = -z_v$ and perpen-

dicular to the z-axis, at (u, v). The mapping, $(\theta, \phi) \rightarrow (u, v)$ is

$$u = z_v \frac{\sin\psi(1 - \beta\cos\theta) + \beta\sin\theta\cos\phi\cos\psi}{-\cos\psi(1 - \beta\cos\theta) + \beta\sin\theta\cos\phi\sin\psi}$$
$$v = z_v \frac{\beta\sin\theta\sin\phi}{-\cos\psi(1 - \beta\cos\theta) + \beta\sin\theta\cos\phi\sin\psi},$$

where $\beta = \frac{R}{\rho}$. Conversely, the inverse mapping $(u, v) \rightarrow (\theta, \phi)$ is

$$\phi = \tan^{-1}\Big(\frac{v}{z_v\sin\psi + u\cos\psi}\Big)$$
$$\theta = \cos^{-1}\Bigg(\frac{B(u,v) \pm \sqrt{\alpha^2(B(u,v)+1) - B(u,v)}}{\alpha(B(u,v)+1)}\Bigg)$$

where $B(u, v) = \dfrac{v^2 + (z_v\sin\psi + u\cos\psi)^2}{(u\sin\psi - z_v\cos\psi)^2}$.

The plus sign in the equation for θ gives the front facing intersection; the negative sign gives the back facing one.

These two (nonlinear) mappings transform points from the surface of the sphere to the view plane and vice versa. They are evaluated frequently during stepping. (Values outside the range of the trigonometric functions indicate view plane coordinates that do not intersect the sphere.)

5.3 Sampling

For planes and spheres the visibility curves map to lines on the view plane. We could follow those lines away from the CoP_v calculating once per pixel traversed, but it is more attractive to combine antialiasing using supersampling with the rendering code. Then we wish to sample along visibility curves so that approximately the same number of samples fall in each pixel.

The following algorithm gives a reasonable uniform density of samples, across the view plane.

1. Choose the number of samples per pixel, n.
2. Set the density of the visibility lines so that they are spaced $1/\sqrt{n}$ pixels apart at the point in the mapped region farthest from the CoP.
3. For each visibility line,
   (a) Place the first sample at the trailing edge of the mapped region. Its distance from CoP_v is r_0.
   (b) Place sample j at r_j from CoP_v where $r_j = r_{j-1} - r_0/r_{j-1}\sqrt{n}$.
   (c) Stop placing samples when a sample is past the leading edge or negative.
   (d) Jitter all the samples along the visibility line, not off it.

For each visibility line the stepping algorithm is then used to find the intersection between the height field and an eye ray constructed through each sample in turn, starting with the sample closest to the CoP_v. Occasionally, stepping is not terminated when the last sample is reached, which occurs at the trailing edge. Then extra samples are generated by inverting the sample generation process and continuing outside the mapped region.

For each pixel the samples that fall within it are averaged. The algorithm described does not result in uniform sampling, since samples have to remain along visibility lines of the view plane to provide the fundamental coherence in our height and ray intersection algorithm.

5.4 Pseudo-code

The stepping algorithm (Section 4), the transformations between the view plane and the underlying surface and the geometry of visibility curves (Section 5.1 and 5.2) contribute to the following implementation.

First, the CoP_v is located from the geometry of the underlying surface and the view position. Then, using the algorithm described in Section 5.3, the samples on the view plane are pre-computed. Finally the stepping algorithm for each visibility line is executed. Rough pseudo-code for an implementation of the stepping algorithm follows.

For each visibility line,

1. Calculate the first solution at the leading edge.
2. While the visibility line is not fully processed:
   (a) Get next inside sample point or create extra sample if beyond trailing edge.
   (b) Make the eye ray for the sample.
   (c) While a close enough solution is not found for the sample:
      i. Step horizontally from previous solution to the eye ray.
      ii. If no height field, visibility curve ends. Process trailing edge solution and reiterate pseudo-code for next line.
      Else step vertically to height field. This is now the solution.
      iii. If horizontal step was smaller than threshold, output solution and goto 2(a).
      Else if horizontal step was a reverse, find reverse solution, output solution and goto 2(a).
      Else goto 2(c).

6 Results

In this section, we present our results and describe how the convergence threshold affects the quality of the renderings and efficiency of the algorithm.

The input height fields used are a scanned height field with accompanying fields of colour and texture normals, as in Figure 8(e), or are generated from one, Figure 8(a, c, d, f), or multiple, Figure 8(b), sinusoidal functions. When using sinusoidal functions with the height encoded as a colour ramp (highest to lowest: red, yellow, green, cyan, blue and then purple), artifacts are quickly noticed, easily diagnosed, and are guaranteed not to be the result of sampling noise. The image with six peaks on a plane, Figure 8(b), is made from adding six Laplacian functions, each having a different amplitude, width and location. This test image demonstrates complex occlusion and a correct silhouette at both leading and trailing edges. The simple sinusoidal image, Figure 8(d), is a good test case for understanding the algorithm and identifying pathologies.

The images in Figures 8 show the capabilities of the stepping algorithm on both underlying surfaces. It handles occlusion, finding the closest intersection when there are multiple intersections, without having to perform 3D computation. Correct trailing edge silhouettes of the displacement map are produced by continuing to find intersections for eye rays that intersect the underlying surface outside the displacement map. The results also demonstrate the resampling and antialiasing described in Section 5.3.

For the plane surface, the images in Figures 8(b, c, d) show low angle eye rays that are challenging to get right, but perceptually important because the silhouette shows prominently the 3D shape of the displacement. When the eye ray intersects the underlying surface at a grazing angle, the algorithm automatically performs the careful silhouette calculation at grazing view angles, especially at the trailing edges of the underlying surface. In addition, at grazing angles there is higher risk of steps passes through narrow features.

	Unslanted		Slanted	
	$\epsilon = 0.25$	$\epsilon = 0.001$	$\epsilon = 0.25$	$\epsilon = 0.001$
Visibility curves	1 079	1 079	1 117	1 117
Inside samples	374 996	374 996	238 706	238 706
Extra samples	37 728	36 087	69 775	66 330
Total samples	412 724	411 083	308 481	305 036
Advance steps	472 559	1 180 941	420 785	602 868
Reverse steps	15	265 689	60 340	231 364
Binary cuts	101	1 219 162	98 944	1 082 847

Table 1: Comparison of the number of operations required for the sine wave displacement map (Figures 8(d), 9–11). These values are computed for 2 samples per pixel, and scale linearly with the number of samples, except for the number of rays which scales as the square root of the number of samples.

To render correctly at these view angles, the convergence threshold, ϵ, which is a fraction of the inter-sample distance, needs to be smaller than when the displacement map is seen at a steep angle.

Figure 9 shows the variations in image quality for different convergence thresholds for the sinusoidal displacement map seen at a grazing angle. The rendering has fewer artifacts when ϵ is smaller. The image on the left of Figure 9 shows artifacts on the leading and trailing edges of the frontal plane, and at the occluding edges, which are especially visible against the black background. In the image on the right, these artifacts are reduced, but a seam of non-red samples is still noticeable along the occlusion line. For Figure 8(d), where $\epsilon = 0.001$ the rendering is clean, the occluding edges are everywhere red. Figure 11 shows close up views of the three images at the trailing edge of the far right corner. Also displayed is the number of steps required for each pixel. The extra iterations when the convergence threshold is decreased are responsible for the improved result.

In contrast with the grazing angle of view, Figure 10 shows that for steep view angles a much larger ϵ provides quick computation of artifact-free images. Ultimately, controlling the step size automatically, depending on height field curvature and the distance between eye rays, will offer the best solution.

Also, the image quality provided by a given convergence threshold, ϵ, depends on the features present in the height map. The left image of Figure 12 shows that for a low view angle a relatively small threshold, $\epsilon = 0.01$, leaves small artifacts on the thinner and higher peaks. The right top picture shows the number of iterations in the middle peak, to contrast with the image in Figure 8(b), where $\epsilon = 0.0001$ and there are no artifacts.

We evaluate the efficiency of the stepping algorithm in terms of the number of basic operations required per sample. For each sample on a visibility curve, two operations, eye ray intersection and height field evaluation, are iterated to find the intersection with the height map. The efficiency of the algorithm depends on the number of evaluations required of these basic operations. Table 1 shows measured values for the sine wave displacement map. The two columns labelled 'Unslanted' gives values when the surface is viewed from a high angle. The extra samples, which are added outside the mapped region to get the silhouette correct, are about 10% of the samples inside the region, which we expect because the silhouette lies close to the edge of the rectangular region. When the threshold is generous, one quarter of the pixel size, only a little more than one advance step is needed per sample; there are almost no reverses and binary search is rare. However, when the threshold is set much smaller, the number of advance steps per sample increases to about three. Equally serious, more than half the samples produce a reverse because advance steps beyond the intersection are much more likely, and four to five binary interpolations are required before the threshold is reached. In total about six evaluations are needed per sample.

When the same surface is viewed from a low angle, (the two columns labelled 'Slanted'), the number of inside samples is smaller because the mapped region covers less of the display, but the number of extra samples is much greater because the silhouette goes farther from the underlying surface. Well over a quarter of the samples are extra. When the threshold is generous, the number of advance steps is scarcely increased compared to the unslanted view, but the number of reverses increases greatly. Reverses occur on a fifth of the samples, but the number of binary interpolations per reverse is small, less than two. This more challenging view less than doubles the number of evaluations required. When the threshold is very small the slanted view performs little worse than the unslanted one, about six to seven evaluations per sample.

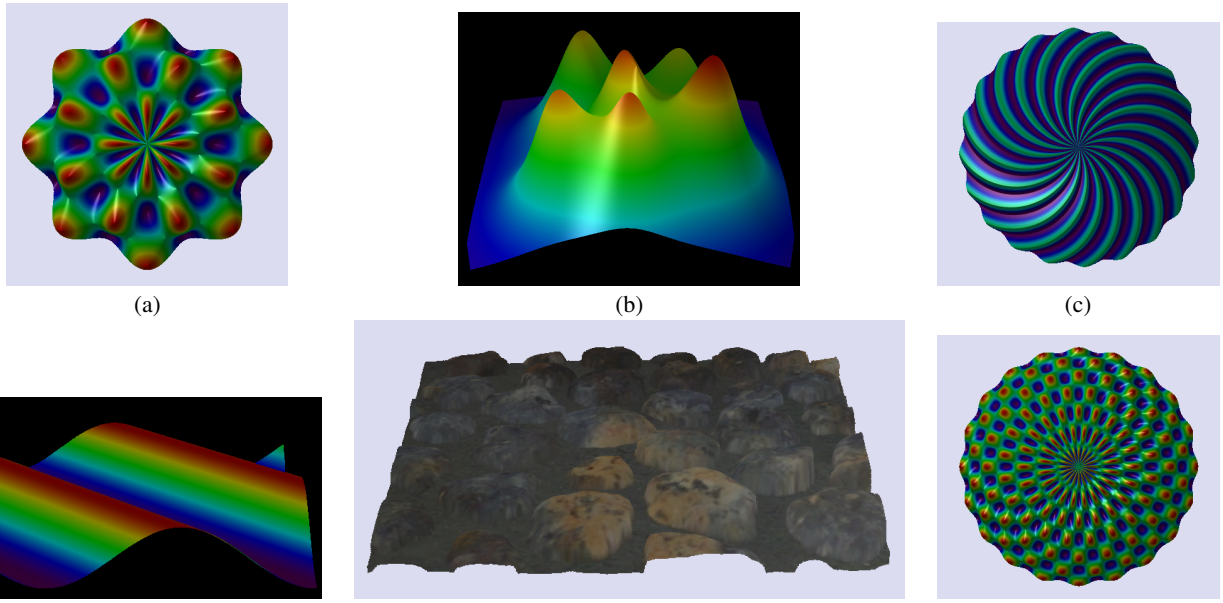
The important difference between the slanted and unslanted views is that the generous threshold produces a good rendering for the unslanted view, while the strict threshold is required to get a good rendering for the unslanted one. Thus, relaxing the threshold without loss of quality for low angle views is the most promising avenue for increasing the efficiency of the algorithm. As for comparing this algorithm with others in terms of efficiency, the result is highly dependent on architecture. The basic evaluations required for stepping, in terms of which we measure efficiency require only a few arithmetic operations on the CPU. Whether or not they can be equally simply implemented on a modern GPU remains for future research.

7 Conclusions

The stepping algorithm for displacement mapping on planes and spheres, which we described in this paper, correctly evaluates self-occlusion and silhouettes, avoids three-dimensional computation, takes advantage of spatial coherence in the height field, minimizes inter-pixel interactions caused by multiple solutions, includes resampling and antialiasing of colours and texture normals in the algorithm, and supports self-shadowing using shadow maps. We measured the efficiency of the algorithm in terms of a basic computational unit, which calculates the position of the eye ray (in one dimension) at a given height above the surface and evaluates the height field at that point. On average, much less than ten such evaluations per sample are required for the more challenging cases. This efficiency is achieved by taking advantage of spatial coherence.

This paper makes three contributions to the geometry of displacement map. The first is the importance of the visibility curves, the underlying surface and the CoPs, especially in the presence of multiple solutions. These concepts provide a good foundation for formalizing the geometry of displacement mapping. The second is the demonstration that for simple underlying surfaces, displacement mapping can be implemented in one dimension. The third contribution, a minor one, is the systematic method of sampling in the presence of visibility curves. The implementation provides these desirable features, while simultaneously resampling the texture map and doing antialiasing.

By keeping the geometry simple and considering only two simple underlying surfaces, we were able to lower the dimension of displacement mapping. In the future we plan to extend the work presented in this paper to calculate the visibility curves on arbitrary manifolds using differential geometry, and thus gain the advantages of our algorithm in generalized displacement mapping. The geometry of general surfaces is more challenging, but the singularities that exist have been well studied [1]. The visibility curves are not, in general, straight lines on the view plane, but the generalization is likely to be important because it enables possible recursive texture mapping as a way of handling multiple levels of detail.



(a) (b) (c) (d) (e) (f)

Figure 8: Images showing the capabilities of the stepping algorithm on both underlying surfaces.

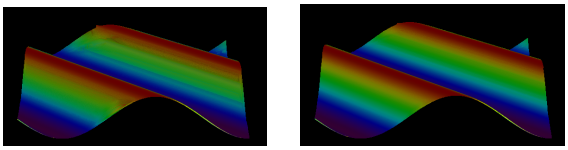


Figure 9: Effect on rendering quality given by the convergence threshold value, ϵ. On the left, $\epsilon = 0.25$ whereas on the right $\epsilon = 0.1$.

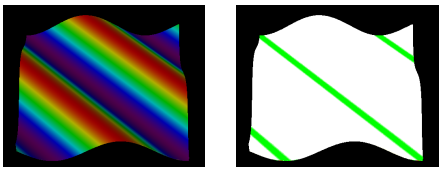


Figure 10: At a steep viewing angle, image produced with $\epsilon = 0.25$ presents no artifacts (left) and used a small number of iterations (on the right, white means 1 iteration and green means $[2, 3]$).

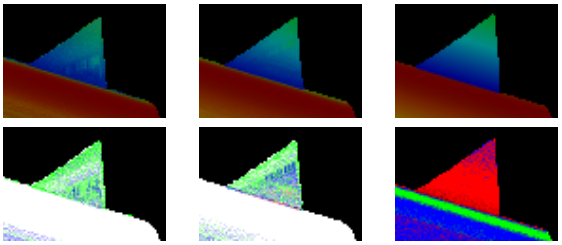


Figure 11: Close up comparison of rendering quality with the number of iterations taken. First row, on the left $\epsilon = 0.25$, on the middle $\epsilon = 0.1$, respective close up of images in Figure 9, and on the right $\epsilon = 0.001$, close up of Figure 8. Second row, iteration numbers are coded by colour. Dark red means more than 8, red $]5, 8]$, blue $]3, 5]$, green $[2, 3]$ iterations, and white means 1 iteration.

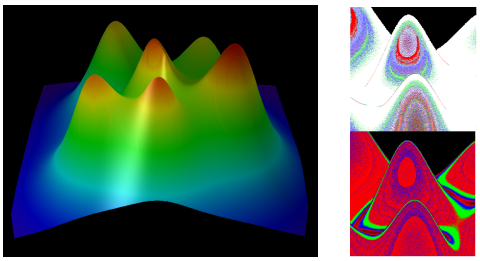


Figure 12: With $\epsilon = 0.01$ on a grazing angle the peaks show artifacts whereas in Figure 8(b) where $\epsilon = 0.0001$ there are no artifacts. The top right picture shows number of iterations for the left image, on the bottom it is the equivalent for Figure 8(b).

ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their comments. All the authors were supported by NSERC grants during the time of this research.

REFERENCES

[1] V. I. Arnold, V. S. Afrajmovich, Y. S. Il'yashenko, and L. P. Shil'nikov. *Bifurcation Theory and Catastrophe Theory*. Springer-Verlag, Berlin, 1999.

[2] J. F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (Proceedings of SIGGRAPH '78)*, volume 12, pages 286–292, Aug. 1978.

[3] R. L. Cook. Shade trees. In *Computer Graphics (Proceedings of SIGGRAPH '84)*, volume 18, pages 223–231, July 1984.

[4] W. Donnelly. Per-pixel displacement mapping with distance functions. In *GPU Gems 2*, chapter 8, pages 123–136. Addison Wesley, Mar. 2005.

[5] J. C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, Dec. 1996.

[6] J. Kautz and H.-P. Seidel. Hardware accelerated displacement mapping for image based rendering. In *Graphics Interface '01*, pages 61–70, 2001.

[7] C.-H. Lee and Y. Shin. A terrain rendering method using vertical ray coherence. *Journal of Visualization & Computer Animation*, 8(2):97–114, Apr. 1997.

[8] R. Leung and S. Mann. Distortion minimization and continuity preservation in surface pasting. In *Graphics Interface '03*, pages 193–200, 2003.

[9] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. In *Computer Graphics (Proceedings of SIGGRAPH '00*, pages 369–374, 2000.

[10] N. L. Max. Vectorized procedural models for natural terrain: Waves and islands in the sunset. In *Computer Graphics (Proceedings of SIGGRAPH '81)*, volume 15, pages 317–324, Aug. 1981.

[11] K. Oh, H. Ki, and C.-H. Lee. Pyramidal displacement mapping: a GPU based artifacts-free ray tracing through an image pyramid. In *Symposium on Virtual Reality Software and Technology '06*, pages 75–82, 2006.

[12] M. Pharr and P. Hanrahan. Geometry caching for ray-tracing displacement maps. In *Eurographics workshop on Rendering techniques '96*, pages 31–40, 1996.

[13] H. Qu, F. Qiu, N. Zhang, A. Kaufman, and M. Wan. Ray tracing height fields. In *Computer Graphics International*, pages 202–209, 2003.

[14] N. Tatarchuk. Dynamic parallax occlusion mapping with approximate soft shadows. In *Symposium on Interactive 3D graphics and games*, pages 63–69, Mar. 2006.

[15] R. D. Toledo, B. Wang, and B. Levy. Geometry textures. In *Brazilian Symposium on Computer Graphics and Image Processing*, pages 79–86, Oct. 2007.